Semi-automatic implementation of the complementary error function

Anastasia Volkova (Intel & Inria)
Jean-Michel Muller (CNRS)

ARITH-26
June 11, 2019

Don't write code, generate it.

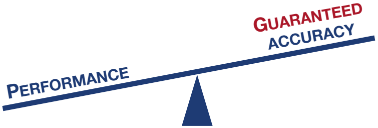
- Mathematical functions are costly
  - → rich trade-off possibilities
- Standard libm is not enough
- A "flavor" per application/target platform
  - → high human resource consumption

Don't write code, generate it.

- Mathematical functions are costly
  → rich trade-off possibilities
- Standard libm is not enough
- A "flavor" per application/target platform
  → high human resource consumption

Our approach:

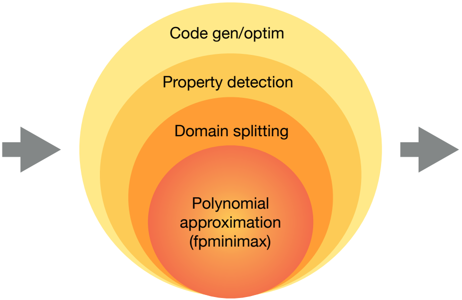
- Automate
- Generate code on-demand
- Adapt for specific context



Metalibm

code generator for libm and beyond

Function
Domain
Target error
...



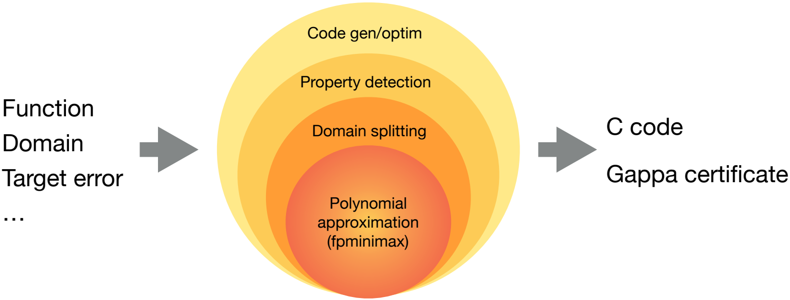
C code

Gappa certificate

- Easy to use
- Performance comparable to handwritten code
- Deals with a variety of elementary functions

Metalibm

code generator for libm and beyond



- Easy to use
- Performance comparable to handwritten code
- Deals with a variety of elementary functions **...but special functions remain a challenge**

Erf and erfc

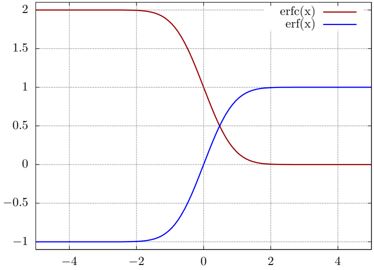
$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$

Quick convergence:

$\circ(\mathrm{erf}(6)) = 1$ in binary64

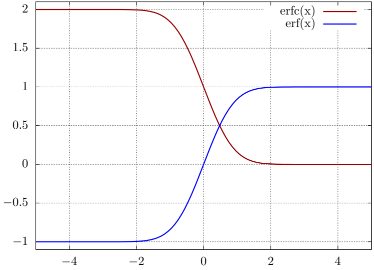
$\circ(\mathrm{erfc}(28)) = 0$ in binary64



Erf and erfc

$$\text{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

$$\text{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$



Quick convergence:

$\circ(\text{erf}(6)) = 1$ in binary64

$\circ(\text{erfc}(28)) = 0$ in binary64

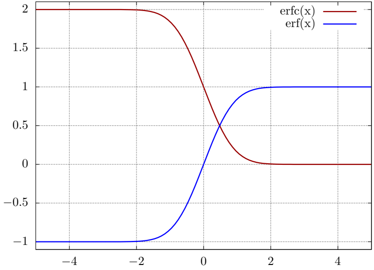
Metalibm with binary64 target accuracy:

- deals with erf(x) on $[0; 6]$ within 49 sec
- fails with erfc(x) on $[0; 28]$ even after 3 h

Erf and erfc

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

$$\mathrm{erfc}(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$$



Quick convergence:

$\circ(\mathrm{erf}(6)) = 1$ in binary64

$\circ(\mathrm{erfc}(28)) = 0$ in binary64

Metalibm with binary64 target accuracy:

- deals with $\mathrm{erf}(x)$ on $[0;6]$ within 49 sec
- fails with $\mathrm{erfc}(x)$ on $[0;28]$ even after 3 h

Reason:

- $\mathrm{erfc}(x)$ is too "flat"
- not close enough to asymptotic expression $e^{-x^2}/(x\sqrt{\pi})$

Code generation for the erfc(x)

Input: relative error bound δ
Output: C code using binary64 data/arithmetic

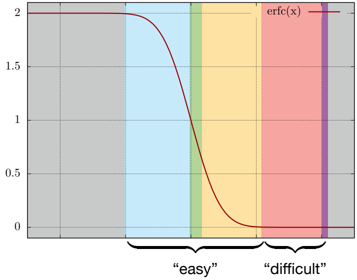
Our approach:

"Easy" zones:

- directly use Metalibm

"Difficult" zone:

- asymptotic expression
- correction back to erfc(x)
- re-partition of the error budget

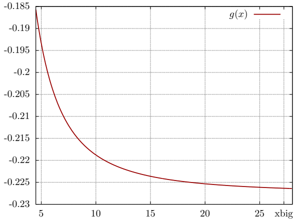


Approximation technique

Easier-to-approximate function:

$$g(x) = \frac{1}{x e^{x^2} \operatorname{erfc}(x)} - 2$$

- decreasing on $[5; x_{\text{BIG}}]$
- $|g(x)| \leq 2 - \sqrt{\pi} \leq 0.228$



Correction:

$$\operatorname{erfc}(x) = \frac{e^{-x^2}}{2x + x g(x)}$$

Evaluation:

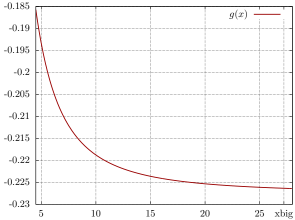
- approximate exp and g
- recover erfc

Approximation technique

Easier-to-approximate function:

$$g(x) = \frac{1}{x e^{x^2} \operatorname{erfc}(x)} - 2$$

- decreasing on $[5; x_{\text{BIG}}]$
- $|g(x)| \leq 2 - \sqrt{\pi} \leq 0.228$



Correction:

$$\operatorname{erfc}(x) = \frac{e^{-x^2}}{2x + x g(x)}$$

Issue:

- $-x^2 \in [-741.256, -25]$
- exp will *underflow*

Evaluation:

- approximate exp and g
- recover erfc

What is the best way to scale?

Choose a scaling s to be within $[-708.396\cdots; 670.96\cdots]$

$$e^{-x^2+s-s}$$

What is the best way to scale?

Choose a scaling s to be within $[-708.396\cdots; 670.96\cdots]$

$$e^{-x^2+k\ln 2}\cdot 2^{-k}$$

What is the best way to scale?

Choose a scaling s to be within $[-708.396\cdots; 670.96\cdots]$

$$e^{-x^2+k\ln 2}\cdot 2^{-k}$$

Search for $k\in\mathbb{Z}$ that minimizes $|s-\circ(k\ln 2)|$

- for FP representation $k=61$, $-x^2+\hat{s}\in[-698.9\cdots,17.2\cdots]$
- for DD representation $k=1021$, $-x^2+\hat{s}\in[-33.5\cdots,682.7\cdots]$

Error analysis and repartition

Task: ensure a relative error δ and deduce accuracy of each step in

$$\operatorname{erfc}(x) = 2^{-k} \frac{e^{-x^2+\hat{s}}}{2x + xg(x)}$$

Error analysis and repartition

Task: ensure a relative error δ and deduce accuracy of each step in

$$\mathrm{erfc}(x) = 2^{-k}\frac{e^{-x^2+\hat{s}}}{2x + xg(x)}$$

$$y(x) = 2^{-k}a(x)/d(x)$$

$$a(x) = e^{t(x)}$$

$$t(x) = -x^2 + \hat{s}$$

$$d(x) = 2x + r(x)$$

$$r(x) = xg(x)$$

Error analysis and repartition

Task: ensure a relative error δ and deduce accuracy of each step in

$$\mathrm{erfc}(x) = 2^{-k}\frac{e^{-x^2+\hat{s}}}{2x + xg(x)}$$

$$\begin{aligned} y(x) &= 2^{-k} a(x)/d(x) \\ a(x) &= e^{t(x)} \\ t(x) &= -x^2 + \hat{s} \\ d(x) &= 2x + r(x) \\ r(x) &= xg(x) \end{aligned}$$

$$\hat{a}(x) = a(x)(1+\varepsilon_a),$$

$$\hat{d}(x) = d(x)/(1+\varepsilon_d)$$

$$\begin{aligned} \hat{y}(x) &= 2^{-k}\frac{a(x)}{d(x)}(1+\varepsilon_{\mathrm{DIV}})(1+\varepsilon_a)(1+\varepsilon_d) \\ &= 2^{-k}\frac{a(x)}{d(x)}(1+\varepsilon_y) \end{aligned}$$

To ensure $\varepsilon_y \le \varepsilon$ it suffices to ensure

$$|\varepsilon_a| \le \varepsilon/4, \quad |\varepsilon_d| \le \varepsilon/4, \quad |\varepsilon_{\mathrm{DIV}}| \le \varepsilon/4.$$

Error analysis and repartition

Task: ensure a relative error δ and deduce accuracy of each step in

$$\mathrm{erfc}(x) = 2^{-k} \frac{e^{-x^2+\hat{s}}}{2x + xg(x)}$$

$$y(x) = 2^{-k} a(x)/d(x)$$
$$a(x) = e^{t(x)}$$
$$t(x) = -x^2 + \hat{s}$$
$$d(x) = 2x + r(x)$$
$$r(x) = xg(x)$$

$$\hat{t}(x) = t(x) + \Delta_t$$

$$\begin{aligned}\hat{a}(x) &= \mathrm{EXP}(t(x) + \Delta_t) = e^{t(x)+\Delta_t}\,(1 + \varepsilon_{\mathrm{EXP}}) \\ &= e^{t(x)}(1 + e^{\Delta_t} - 1)\,(1 + \varepsilon_{\mathrm{EXP}}) \\ &= e^{t(x)}\,(1 + \varepsilon_a)\,,\end{aligned}$$

To ensure $\varepsilon_a \leq \varepsilon$ it suffices to ensure

$$|\varepsilon_{\mathrm{EXP}}| \leq \varepsilon/4, \quad |\Delta_t| \leq \ln(1 + \varepsilon/4)$$

Generic error bounds

Computation step	Error	Examples of error requirements		
	ε_y	δ	2^{-32}	2^{-46}
$y(x) = 2^{-k}a(x)/d(x)$	ε_{DIV}	$\delta/4$	2^{-34}	2^{-48}
$a(x) = e^{t(x)}$	ε_{EXP}	$\delta/16$	2^{-36}	2^{-50}
$t(x) = -x^2 + k\ln 2$	Δ_t	$\ln(1+\delta/16)$	$1.99 \cdot 2^{-37}$	$1.99 \cdot 2^{-51}$
$d(x) = 2x + r(x)$	ε_{ADD}	$\delta/8$	2^{-35}	2^{-49}
$r(x) = xg(x)$	ε_{MUL}	$\frac{\delta}{4\overline{\alpha}(8+\delta)}$	$1.94 \cdot 2^{-35}$	$1.94 \cdot 2^{-49}$
$g(x)$	ε_g	$\frac{\delta}{4\overline{\alpha}(8+\delta)}$	$1.94 \cdot 2^{-35}$	$1.94 \cdot 2^{-49}$

Generic error bounds

Computation step	Error	Examples of error requirements		
	ε_y	δ	2^{-32}	2^{-46}
$y(x) = 2^{-k} a(x)/d(x)$	ε_{DIV}	$\delta/4$	2^{-34}	2^{-48}
$a(x) = e^{t(x)}$	ε_{EXP}	$\delta/16$	2^{-36}	2^{-50}
$t(x) = -x^2 + k \ln 2$	Δ_t	$\ln(1 + \delta/16)$	$1.99 \cdot 2^{-37}$	$1.99 \cdot 2^{-51}$
$d(x) = 2x + r(x)$	ε_{ADD}	$\delta/8$	2^{-35}	2^{-49}
$r(x) = xg(x)$	ε_{MUL}	$\frac{\delta}{4\overline{\alpha}(8+\delta)}$	$1.94 \cdot 2^{-35}$	$1.94 \cdot 2^{-49}$
$g(x)$	ε_g	$\frac{\delta}{4\overline{\alpha}(8+\delta)}$	$1.94 \cdot 2^{-35}$	$1.94 \cdot 2^{-49}$

. . . but what happens in double precision?

When straightforward binary64 is used

Computation step	Error	Bounds
	$\|\varepsilon_y\|$	δ
$y(x) = 2^{-k}a(x)/d(x)$	$\|\varepsilon_{\mathsf{DIV}}\|$	u
$a(x) = e^{t(x)}$	$\|\varepsilon_{\mathsf{EXP}}\|$	$1.\cdots\delta - 1024.2584u$
$t(x) = -x^2 + k\ln 2$	$\|\Delta_t\|$	$1024.2583u$
$d(x) = 2x + r(x)$	$\|\varepsilon_{\mathsf{ADD}}\|$	u
$r(x) = xg(x)$	$\|\varepsilon_{\mathsf{MUL}}\|$	u
$g(x)$	$\|\varepsilon_g\|$	$1.7\delta - 9.6u$

When straightforward binary64 is used

Computation step	Error	Bounds
	$\lvert\varepsilon_y\rvert$	δ
$y(x) = 2^{-k} a(x)/d(x)$	$\lvert\varepsilon_{\mathrm{DIV}}\rvert$	u
$a(x) = e^{t(x)}$	$\lvert\varepsilon_{\mathrm{EXP}}\rvert$	$1.\cdots\delta - 1024.2584u$
$t(x) = -x^2 + k\ln 2$	$\lvert\Delta_t\rvert$	$1024.2583u$
$d(x) = 2x + r(x)$	$\lvert\varepsilon_{\mathrm{ADD}}\rvert$	u
$r(x) = xg(x)$	$\lvert\varepsilon_{\mathrm{MUL}}\rvert$	u
$g(x)$	$\lvert\varepsilon_g\rvert$	$1.7\delta - 9.6u$

- Arithmetic with relative error u

When straightforward binary64 is used

Computation step	Error	Bounds
	$\lvert\varepsilon_y\rvert$	δ
$y(x) = 2^{-k}a(x)/d(x)$	$\lvert\varepsilon_{\text{DIV}}\rvert$	u
$a(x) = e^{t(x)}$	$\lvert\varepsilon_{\text{EXP}}\rvert$	$1.\cdots\delta - 1024.2584u$
$t(x) = -x^2 + k\ln 2$	$\lvert\Delta_t\rvert$	$1024.2583u$
$d(x) = 2x + r(x)$	$\lvert\varepsilon_{\text{ADD}}\rvert$	u
$r(x) = xg(x)$	$\lvert\varepsilon_{\text{MUL}}\rvert$	u
$g(x)$	$\lvert\varepsilon_g\rvert$	$1.7\delta - 9.6u$

- Arithmetic with relative error u
- Can adapt only the accuracy of $exp(x)$ and $g(x)$

When straightforward binary64 is used

Computation step	Error	Bounds
	$\lvert\varepsilon_y\rvert$	δ
$y(x) = 2^{-k}a(x)/d(x)$	$\lvert\varepsilon_{\text{DIV}}\rvert$	u
$a(x) = e^{t(x)}$	$\lvert\varepsilon_{\text{EXP}}\rvert$	$1.\cdots\delta - 1024.2584u$
$t(x) = -x^2 + k\ln 2$	$\lvert\Delta_t\rvert$	$1024.2583u$
$d(x) = 2x + r(x)$	$\lvert\varepsilon_{\text{ADD}}\rvert$	u
$r(x) = xg(x)$	$\lvert\varepsilon_{\text{MUL}}\rvert$	u
$g(x)$	$\lvert\varepsilon_g\rvert$	$1.7\delta - 9.6u$

- Arithmetic with relative error u
- Can adapt only the accuracy of $exp(x)$ and $g(x)$
- **Restriction on the relative error:** $\delta > 5.002 \cdot 2^{-38}$

When straightforward binary64 is used

Computation step	Error	Bounds
	$\lvert\varepsilon_y\rvert$	δ
$y(x) = 2^{-k} a(x)/d(x)$	$\lvert\varepsilon_{\text{DIV}}\rvert$	u
$a(x) = e^{t(x)}$	$\lvert\varepsilon_{\text{EXP}}\rvert$	$1.\cdots\delta - 1024.2584u$
$t(x) = -x^2 + k \ln 2$	$\lvert\Delta_t\rvert$	$1024.2583u$
$d(x) = 2x + r(x)$	$\lvert\varepsilon_{\text{ADD}}\rvert$	u
$r(x) = xg(x)$	$\lvert\varepsilon_{\text{MUL}}\rvert$	u
$g(x)$	$\lvert\varepsilon_g\rvert$	$1.7\delta - 9.6u$

- Arithmetic with relative error u
- Can adapt only the accuracy of $exp(x)$ and $g(x)$
- **Restriction on the relative error:** $\delta > 5.002 \cdot 2^{-38}$

Must be more accurate in critical parts

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert \Delta_t \rvert \leq 32.259u$	$\lvert \Delta_t \rvert \leq 0.2584u$
6 FP operations	10 FP operations

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$e^{t_h} e^{\Delta_t} e^{t_\ell}$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$e^{t_h} e^{t_\ell} e^{\Delta_t}$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$\lvert\varepsilon_t\rvert \leq 0.2585u$

$e^{t_h} e^{t_\ell} (1 + \varepsilon_t)$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$|\varepsilon_t| \leq 0.2585u$$

$$e^{t_h} e^{t_\ell} (1 + \varepsilon_t)$$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$|\varepsilon_t| \leq 0.2585u$$

$$e^{t_h}(1 + t_\ell)(1 + \varepsilon_t)$$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)(1 + \varepsilon_{E_\ell})(1 + \varepsilon_t)$$

$$|\varepsilon_t| \leq 0.2585u$$
$$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	**Method 2**
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)(1 + \varepsilon_{E_\ell})(1 + \varepsilon_{\text{FMA}})(1 + \varepsilon_t)$$

$$|\varepsilon_t| \leq 0.2585u$$
$$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$$
$$|\varepsilon_{\text{FMA}}| \leq u$$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\|\Delta_t\| \leq 32.259u$	$\|\Delta_t\| \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)(1 + \varepsilon_{E_\ell})(1 + \varepsilon_{\mathrm{FMA}})(1 + \varepsilon_t)$$

$|\varepsilon_t| \leq 0.2585u$

$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$

$|\varepsilon_{\mathrm{FMA}}| \leq u$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)(1 + 1.259u)$$

$|\varepsilon_t| \leq 0.2585u$

$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$

$|\varepsilon_{\text{FMA}}| \leq u$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)(1 + 1.259u)(1 + \varepsilon_{E_h})$$

$$|\varepsilon_t| \leq 0.2585u$$
$$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$$
$$|\varepsilon_{\text{FMA}}| \leq u$$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	Method 2
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)\underbrace{(1 + 1.259u)(1 + \varepsilon_{E_h})}_{(1+\varepsilon_a)}$$

$$|\varepsilon_t| \leq 0.2585u$$
$$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$$
$$|\varepsilon_{\text{FMA}}| \leq u$$

Exploiting double-word arithmetic

- Evaluate $t(x)$ as a double-word $t_h + t_\ell$

Method 1	**Method 2**
$\lvert\Delta_t\rvert \leq 32.259u$	$\lvert\Delta_t\rvert \leq 0.2584u$
6 FP operations	10 FP operations

- Evaluate exponential: $a(x) = e^{t_h} e^{t_\ell} e^{\Delta_t}$

$$e^{t_h}(1 + t_\ell)\underbrace{(1 + 1.259u)(1 + \varepsilon_{E_h})}_{(1+\varepsilon_a)}$$

$$|\varepsilon_t| \leq 0.2585u$$
$$|\varepsilon_{E_\ell}| \leq (1089 \cdot 2^{44})u$$
$$|\varepsilon_{\text{FMA}}| \leq u$$

Result: can approximate with error up to $\delta = 0.76 \cdot 2^{-50}$
Cost: 13 FP operations

Numerical results – 1

Implementation:

- Semi-automatic approximation choice for Metalibm
- Code generation in C

Testing:

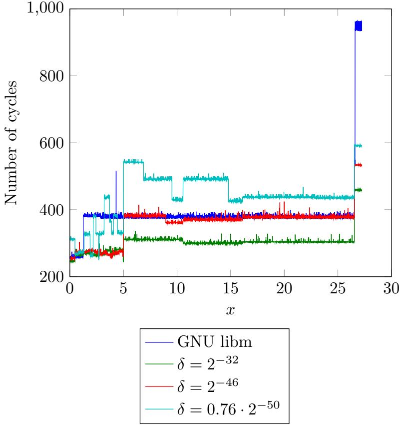
- Reference implementation: GNU libm with gcc 6.3.0
- Target accuracy: $2^{-32}, 2^{-46}, 0.76 \cdot 2^{-50}$

Results:

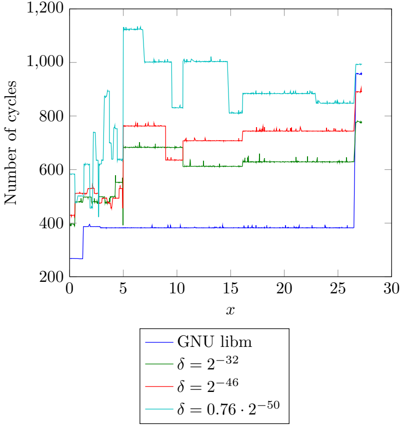
accuracy	$[0; 5]$ abs	$[0; 5]$ rel	$[5; x_{\text{LARGE}}]$ abs	$[5; x_{\text{LARGE}}]$ rel	$[x_{\text{LARGE}}, x_{\text{BIG}}]$ abs
GNU libm	4 ulp	6.34 u	3 ulp	3.98 u	1.5 ulp
$0.76 \cdot 2^{-50}$	2 ulp	3.84 u	4 ulp	4.02 u	1.5 ulp
2^{-46}	18 ulp	21.07 u	15 ulp	16.6 u	1.5 ulp

Numerical results – 2

-O3



-O0



Conclusion

- Partly-automated implementation that offers
  - a priori target accuracy
  - guaranteed error bounds
  - exploration of a large design space
- Asymptotic expression + correction
- Double-word arithmetic for critical parts when in binary64

Perspectives

- Optimize error budget repartition
- Achieve higher accuracy
- Adapt for other functions with similar behavior, e.g. Gaussian